Programmation C++

ING2 – MF

Pourquoi C++ ?

- Votre langage préféré ?

- https://www.tiobe.com/tiobe-index/
- https://www.codementor.io/learn-programming/beginner-programming-language-job-salary-community
- http://spectrum.ieee.org/static/interactive-the-top-programming-languages-2017
- http://www.codingdojo.com/blog/9-most-in-demand-programming-languages-of-2017/

Plan du cours

- 1. Introduction au langage C++ : 1 séance
  - Éléments syntaxiques complémentaires du C
  - Implémentation d'un programme en C++ (fichiers header/code)
- 2. Classe : 2 séances
  - Constructeurs et destructeurs
  - Allocation dynamique : new et delete
- 3. Surcharge des opérateurs : 1 séance
- 4. Héritage et polymorphisme : 2 séances
  - Classe abstraite
  - Fonction virtuelle
  - Redéfinition
- 5. Gestion des flux I/O : 1 séance
- 6. Templates et Librairie de templates STL : 1 séance

C++ versus C

- C est un sous-ensemble de C++ ?
- C++ :
  - multi-paradigme (procédural + orienté-objet)
  - programmation générique
  - type checking plus strict

C++ versus Java

	C++	Java
But	Efficacité d'exécution (performance)	Productivité du programmeur (portabilité)
Liberté	Faire confiance au programmeur	Imposer certaines contraintes
Paradigme de programmation	Procédurale et Orientée objet	Orientée objet
Gestion de mémoire	Manuellement, attention aux fuites mémoires	Garbage collection
Runtime	Compilé en code machine => exécuté par OS Buffer overflows, segmentation faults, …	Compilé en byte code puis interprété par JVM Exceptions
Performance	Compilation statique => code machine optimisé	Byte code mais progrès avec JIT

Références

- **Bjarne Stroustrup, A Tour of C++, 2013**
- Scott Meyers, Effective C++
- …

Séance 1

Introduction au C++

Partie 1 - Découverte de la programmation en C++

Qu'est-ce que le C++ ?

- Très répandu.
- Rapide
- Portable
- Nombreuses bibliothèques
- Multi-paradigmes

Quelques défauts

- Complexité
- Contrôle sur le fonctionnement
- Gestion de la mémoire

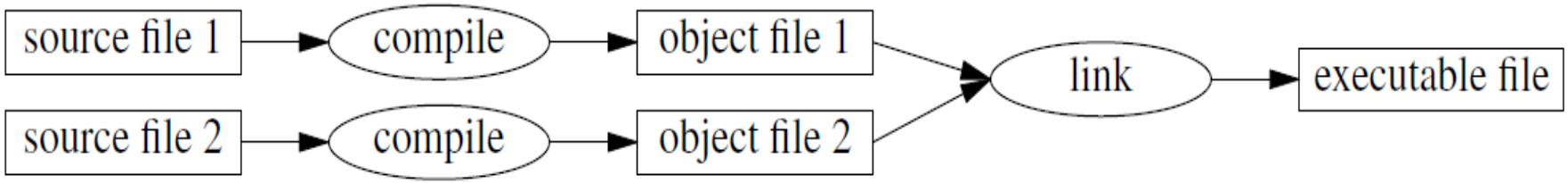
Outils nécessaires pour programmer

- Editeur de texte
- Compilateur
- Débugger (Débogueur en français)

➔3 en 1 **IDE** (ou en français « EDI » pour « Environnement de Développement Intégré »)

➔Projet (plusieurs fichiers de code source : des fichiers.cpp,.h, les images du programme, etc.)

Un projet C++



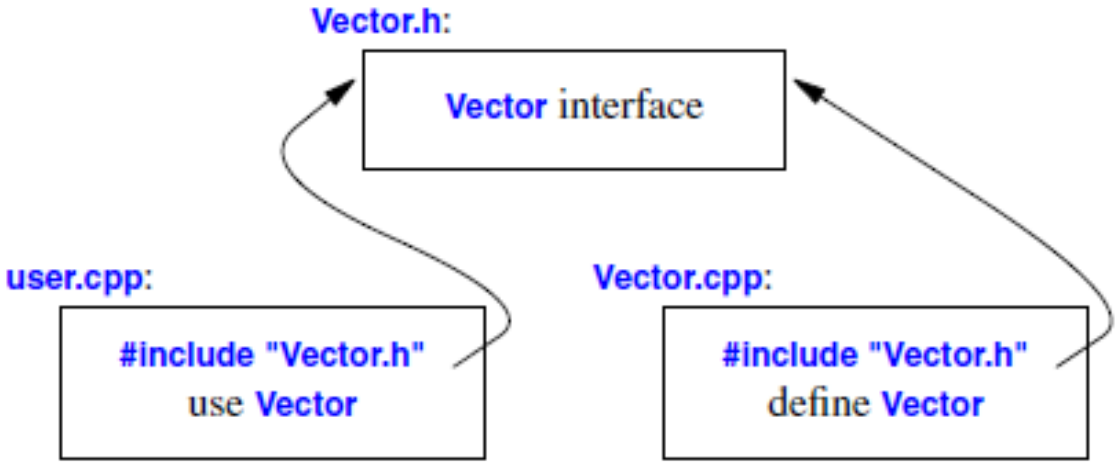
- Fichier d'objet : file.o
- Librairies :
  - file.lib
  - file.dll
- Compilateurs : c++, g++, …

```
g++ -Wall -std=c++11 -c file.cpp
g++ -o prog file1.o file2.o
```

- makefile

Un projet C++

- Fichier d'implémentation : file.cpp, file.cc, file.C
- Fichier de déclaration (header file) : file.h, file.hpp



Premier programme en C++

```cpp
#include <iostream> // Inclut la bibliothèque iostream (affichage de texte)

using namespace std; // Indique quel espace de noms on va utiliser

/*
Fonction principale "main"
Tous les programmes commencent par la fonction main
*/
int main()
{
    cout << "Hello world!" << endl; // Affiche un message
    return 0; // Termine la fonction main et donc le programme
}
```

Premier programme en C++

```cpp
#include <iostream>
using namespace std;

int main()
{
  cout << "Je fais des tests pour apprendre le C++ !" << endl;
  cout << "\"" << endl;
  cout << "\\" << endl;
  return 0;
}
```

les variables

- Nom (Convention)
  - commencent par une minuscule ;
  - décompose en plusieurs mots collés ;
  - chaque nouveau mot (excepté le premier) commence par une majuscule (tauxOccupationSol)
- Valeur
- Type
  - Espace
  - Operateur
- Exemple : TYPE NOM (VALEUR); en C++

  TYPE NOM = VALEUR; en C

Conventions de codage

- Objectifs :
  - Réutilisation de code, maintenance, portabilité
  - Optimisations
  - Éviction des erreurs
  - Sécurité
  - International Obfuscated Code Competition ☺
- Quelques règles :
  - 25 lignes 80 colonnes
  - Indentation
  - Accolades
  - Commentaires
  - Nom de variable : English, cohérent (numChars vs num_chars)
  - Nom de fonction : verbe, CheckForError() vs ErrorCheck(), dump_data_to_file() vs data_file()

Petit test

1. i
2. numberOfCharacters
3. do_this()
4. g()
5. number_of_chars
6. if
7. number of characters
8. get_number_of_characters()
9. is_char_limit()
10. charMax()
11. num1
12. 4nums
13. hxq
14. _num
15. num__chars
16. main
17. cout

Petit test

1. i
2. numberOfCharacters
3. do_this()
4. g()
5. number_of_chars
6. if
7. number of characters
8. get_number_of_characters()
9. is_char_limit()
10. charMax()
11. num1
12. 4nums
13. hxq
14. _num
15. num__chars
16. main
17. cout

Quelle version ?

```
int area_ftoi(float a, float b) { return (int) a * b / 2; }


int iTriangleArea(float fBase, float fHeight) {
    return (int) fBase * fHeight / 2;
}
```

- Nouvelles fonctionnalités hors objet

–le contrôle de type
–les arguments par défaut
–la surcharge de fonctions
–les fonctions inline

Quelques spécificités du C++ (1) hors objet

- **Même type qu'en C**
  - Int, char, float, double,…
- **Existance d'un booléen : bool**
  - True, false
- **Type auto**

```
auto ch = 'x';
auto z = sqrt(y);
```

Quelques spécificités du C++ (1) hors objet

- **inline**

```
inline int Max(int i, int j) {
        if (i>j)
                return i;
        else
                return j;
}
```

  - Pas de récursivité, pas de pointeur
  - Petite fonction, code rapide (boucle)
- **const**
- **volatile**

Quelques spécificités du C++ (1) hors objet

- **Initialisation { } :**

```
double  d1 = 2.3;
double d2 {2.3};
string eisti {”POSE”};

int i = 7.2;          // OK
int i {7.2};          // error
```

Exemple

```
#include <iostream>
using namespace std;

int main()
{
   int ageUser(18);
   int nombreFollowers(432);      //Le nombre d'amis qui te suivent

   double pi(3.14159);

   bool estMonAmi(true);    //Cet utilisateur est-il mon ami ?

   char lettre('a');

   return 0;
}
```

Chaine de caractères

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string nomUser ("leVolontaire");
    return 0;
}
```

Références (ou alias) &

- on utilise le symbole(&) pour déclarer une référence sur une variable
- Exemple
  - int prixRevient(16); //Déclaration d'une variable.
  - int& prixAchat(prixRevient);
- On dit que prixAchat fait référence à prixRevient
- La référence doit impérativement être du même type que la variable qui est référencée

Quelques spécificités du C++

- **Pointeur vs Référence**
  - Code plus clair, plus sûr
    - Poniteur

```
int i=0;
int *pi=&i;
*pi=*pi+1; // Manipulation de i via pi
```

    - Référence

```
int i=0;
int & ri=i; // ri est un identificateur synonyme de i
ri=ri+1; // Manipulation de i via ri
```

  - Passage de paramètre par référence : plus efficace

```
void test(int &i) {
    i = 2; // Modifie le paramètre passé en référence.
    return;
}
```

Quelques spécificités du C++ (1) hors objet

- **Quantificateur const**
  - rend impossible la modification de la variable
  - évite de modifier des données par erreur, utile pour l'optimisation

```
const int cst1 = 4 2 ;
const char * fonction1 ( )  { return "Du texe " ;}
void fonction2 ( const int  tab ) {. . . }
void methodeClasse ( . . . ) const { . . . }
```

- **Volatile :** pas d'optimisation de code par le compilateur

Petit test

- const int *pi[12];

- void (*pf)(int * const pi);

- const char *pc="Coucou !";
- char *pc="Coucou !";

Petit test

- const int *pi[12];

  pi est un tableau de 12 pointeurs, chaque pointeur pointe sur un entier constant

- void (*pf)(int * const pi);

  pf est un pointeur de fonction avec un seul paramètre pi qui est une constante de type pointeur d'entier, et cette fonction ne renvoie rien

- const char *pc="Coucou !";
- char *pc="Coucou !";

Quelques spécificités du C++
namespace

```
#include <iostream>
namespace first {
        int first1;
        int x;
}
namespace second {
        int second1;
        int x;
}
namespace first {
        int first2;
}
int main(){
        //first1 = 1;
        first::first1 = 1;
        using namespace first;
        first1 = 1;
        x = 1;
        second::x = 1;
        using namespace second;
        //x = 1;
        first::x = 1;
        second::x = 1;
        first2 = 1;
        //cout << 'X';
        std::cout << 'X';
        using namespace std;
        cout << 'X';
        return 0;
}
```

Quelques spécificités du C++
Librairie standard d'entrées/sorties

- C++ a redéfini les fonctions d'entrées/sorties pour simplifier leur utilisation
- Il faut inclure le fichier de prototypage suivant : #include <iostream>
- les variables globales stdin, stout et stderr sont remplacées respectivement par cin, cout et cerr.
- l'opérateur << remplace la fonction printf.
- l'opérateur >> remplace la fonction scanf.
- Cette librairie a été définie à l'intérieur de l'espace de noms std. Donc une fonction f s'appele en réalité std :: f. Il faut utiliser la clause using namespace std; pour éviter de mettre le préfixe std ::.

Quelques spécificités du C++
Librairie standard d'entrées/sorties

```
#include <iostream>
using namespace std;
int main() {
int a,b;

cout << "saisir a";
cin >> a;
cout << "le carré de " << a << " vaut " << a*a << endl;
}
```

Classe C++ : Une classe très simple

Le fichier point.h

```
#ifndef _POINT_HPP_
#define _POINT_HPP_
class Point {
        private :
        int x ;
        int y;

        public :
        void setX(int);
        void setY(int);
        int getX();
        int getY();
};
#endif
```

Le fichier point.cpp

```
#include "point.h"
void Point :: setX(int px) {
x = px;
}
void Point :: setY(int py) {
y = py;
}
int Point :: getX() {
return x;
}
int Point :: getY() {
return y;
}
```

Le fichier main.cpp

```
#include <iostream>
#include "point.h"
using namespace std;
int main() {
int a,b;
Point p;
cout << "saisir x";
cin >> a;
cout << "saisir y";
cin >> b;
p.setX(a);
p.setY(b)
cout << " x = " << p.getX() << " y =
" << p.getY();
return 0;
}
```

Classe C++ : Une classe très simple : Constructeur

Le fichier point.h

```
#ifndef _POINT_HPP_
#define _POINT_HPP_
class Point {
    private :
    int x ;
    int y;

    public :
    Point (int px, int py);
    int getX();
    int getY();
};
#endif
```

Le fichier point.cpp

```
#include "point.h"
Point :: Point(int px, int py)
{
x = px;
Y=py;
}
int Point :: getX() {
return x;
}
int Point :: getY() {
return y;
}
```

Le fichier main.cpp

```
#include <iostream>
#include "point.h"
using namespace std;
int main() {
int a,b;
cout << "saisir x";
cin >> a;
cout << "saisir y";
cin >> b;
Point p(a,b);
cout << " x = " << p.getX() <<
" y = " << p.getY();
return 0;
}
```